Atypon Training Program 2023

CONTAINERIZED MICROSERVICES DATA COLLECTION AND ANALYTICS SYSTEM APPLICATIONS

BY: YUSSIF ABDALLA, The University of Jordan

I. ABSTRACT

This report illustrates the solution of the Atypon's 2023 assignment. The assignment is based on the Docker's Engine Containerization technology. A containerized microservices data collection and analytics system is built using a Docker file for each service, and a Docker compose file run the system. The system microservices should perform the specific functions; such as; Enter Data Service, Authentication Service, MySQL Database Service, Mongo Database Service, and Show Results Service.. That is the complex application may be broken up into a series of smaller, more specialized services, each with its own database and its own business logic. Consequently, the microservices may communicate with each other across common interfaces.

Docker Engine is used successfully to deploy the application in an efficient and secure manner in the form of containers. Packaging of the software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable that runs consistently on any infrastructure was possible with using Docker's Engine generated Containers.

II. INTRODUCTION

A newly developed Technology, Docker Engine 2013, that enables developers to deploy applications in an efficient and secure manner is available in the form of containers. Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable—called a container—that runs consistently on any infrastructure. Containers Technology with their lower overhead surpassed the popular virtual machines (VMs), consequently, containers have become the de facto compute units of modern cloud-native applications [1][2].

Containerization eliminates compatibility problems and issues by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run. This single package of software or "container" is independent from the host operating system (i.e. portable) and may run across any platform or cloud, free of issues [1][2].

Nowadays, the software industry adopted microservices for their application development and management in contrast to the single application model. That is a complex application may be broken up into a series of smaller, more specialized services, each with its own database and its own business logic. Then Microservices may communicate with each other across common interfaces (like APIs) and REST interfaces (like HTTP). This modular design enables of working on the specific pieces without impacting the whole [1].

In this work assignment, a containerized microservices data collection and analytics system is built using a Docker file for each service, and a Docker compose file to run the system. The system microservices should perform the following services [3]: Enter Data Service, Authentication Service, MySQL Database Service, Mongo Database Service, and Show Results Service.

III. THE MICROSERVICES SYSTEM DESIGN

3.1 Microservice System Diagram

The microservices system comprises distinct data collection and analytics systems that interact with one another, which is illustrated in Figure 1.

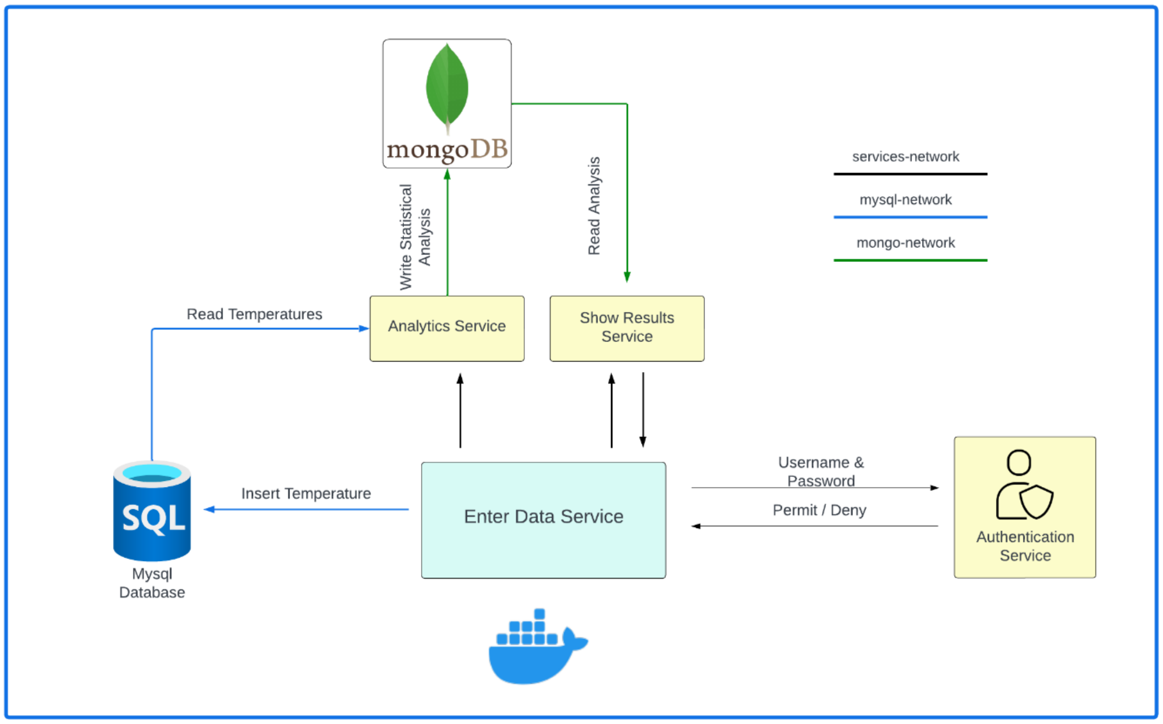

Figure 1: The Microservice System Diagram

These are the primary services that make up the system. These services collaborate to ensure the system functions smoothly, allowing different parts to work together. The main services include:

- **Enter Data Service**
  The data input service serves the purpose of gathering temperature information from users. However, users are granted the ability to input data only after their credentials have been authenticated through the authentication service. Once a user's credentials are successfully verified, they gain the privilege to enter data, which is subsequently recorded within the MySQL Database service.

- **Authentication Service**
  The authentication service plays a crucial role in validating user credentials, which are provided by the enter data service. When the enter data service submits a user's username and password, the authentication service undertakes the process of verification. Upon successful validation, the authentication service sends an authentication response to the enter data service, granting permission for the user to access the data entry page. Conversely, in the event of an unsuccessful validation, an authentication response indicating the authentication failure is transmitted. In such cases, the enter data service redirects the user back to the login page.

- **MySQL Database Service**

  The MySQL Database Service employs a MySQL image to function as its database, primarily dedicated to storing and overseeing temperature-related data. This service serves the purpose of storing temperature readings submitted by users via the enter data service. Furthermore, the analytics service also utilizes this database to retrieve all temperature records and generate insightful statistical analyses.

- **Analytics Service:**

  The Analytics Service retrieves all temperature data from the MySQL database and generates statistical analyses, including metrics like minimum temperature, maximum temperature, and average temperature. Once the analytics are generated, the service uses a MongoDB service to store the resulting analytical information.

- **Mongo Database Service**

  The MongoDB Database Service utilizes a MongoDB image as its database, primarily designed for storing and managing temperature data analytics. This service is focused on storing temperature statistics that are generated by the analytics service. Additionally, the "show results" service also makes use of this database to retrieve all the statistical analyses.

- **Show Results Service**

  The Show Results Service retrieves temperature statistics from the MongoDB database service and presents them to authorized users.

3.2 System Services Dockerfile

All of the system's Dockerfiles for various services share a similar structure. Here are the commands used in these Dockerfiles:

From openjdk:20: This line specifies the base image for the services, utilizing the OpenJDK version 20 image.

COPY ./target/SpringApplication-0.0.1-SNAPSHOT.jar SpringApplication-0.0.1-SNAPSHOT.jar: This command facilitates the transfer of the JAR file from the local file system into the root directory of the Docker container.

CMD ["java", "-jar", "SpringApplication-0.0.1-SNAPSHOT.jar"]: This launches the spring boot application contained in the JAR file

For a comprehensive review of the controller implementation, please refer toAppendix A.

3.3 Microservice Docker Compose File

The docker-compose.yml file serves to define and distribute multi-container applications. It facilitates the execution of images and establishment of networks, enabling seamless communication between services. Figure 2 summarizes the docker-compose configuration for the mentioned system.

```yaml
version: "3.8"
services:
  enter-data-service:
    image: enter-data-service
    ports:
      - "8080:8080"
    networks:
      - mysql-network
      - services-network
    depends_on:
      - mysqldb
      - authentication-service

  authentication-service:
    image: authentication-service
    ports:
      - "8081:8081"
    networks:
      - services-network

  analytics-service:
    image: analytics-service
    ports:
      - "8083:8083"
    networks:
      - mongo-network
      - mysql-network
    depends_on:
      - mysqldb
      - mongodb
```

```
Show-results-service:
    image: show-results-service
    ports:
      - "8084:8084"
    networks:
      - mongo-network
      - services-network
    depends_on:
      - enter-data-service
      - mongodb

mysqldb:
    image: mysql:8
    networks:
      - mysql-network
    ports:
      - "3307:3306"
    volumes:
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: temperature

mongodb:
  image: mongo
  networks:
    - mongo-network
  ports:
    - "27017:27017"


networks:
  services-network:
  mysql-network:
  mongo-network:
```

Figure 2: The docker-compose.yml File

3.4 Microservice System Connections and Networking

In the microservices system, three distinct networks are utilized. The first network, known as "mysql-network," facilitates communication with the MySQL database. The second network, referred to as "mongo-network," is responsible for handling communication with MongoDB. The third network, named "services-network," is exclusively dedicated to inter-service communication.

By implementing dedicated networks for the databases, a strong level of isolation is achieved between the database containers and other containers within the application. This isolation significantly enhances security by limiting access solely to the essential containers or hosts.

The "mysql-network" serves the purpose of accessing the database by services, such as adding temperature data to the temperature tables. This functionality is carried out by the enter data service.

Similarly, the "mongo-network" plays a crucial role in executing MongoDB operations, particularly in retrieving statistical insights about temperatures. This task is performed by the show results service.

The "services-network" enables the enter data service to connect to the authentication service, situated on the same network, for the purpose of user validation. Additionally, the "services-network" facilitates a connection between the enter data service and the show results service, enabling the visualization of statistical temperature data.

Every service within the system is configured to listen on a specific port. In the docker-compose file, both a host port and a container port are defined for each service. These port assignments, as specified in the docker-compose file, are then shared among the various services outlined in the docker-compose configuration. Table (1) illustrates the host and container ports that are used for different services.

Table 1: System Services Port Assignment

Service Name	Ports (Host:Container)
`enter-data-service`	`8080:8080`
`authentication-service`	`8081:8081`
`analytics-service`	`8083:8083`
`show-results-service`	`8084:8084`
`mysqldb`	`3307:3306`
`mongodb`	`27017:27017`

IV. THE MICROSERVICES SYSTEM IMPLEMENTATION

In the system's implementation, several web applications were built using the Spring framework. These applications utilize both Spring MVC and Spring REST implementations. Spring MVC is employed for web applications responsible for rendering views within the system, such as the “enter data service”. These applications generate and deliver dynamic HTML views to users, facilitating interactions. On the other hand, Spring REST is used for web applications that handle requests related to database operations, like the “analytics service”. These applications provide APIs that allow clients to communicate with the server for data-related tasks.

To demonstrate all the different services and their implementations the subsequent paragraphs becomes handy.

Enter Data Service

The enter data service uses Spring MVC for its implementation. This web application consists of three different controllers.

1. **The Login Controller**

The Login Controller is responsible for displaying the main login page to the user. It also handles the submission of the user's credentials when they click the submit button. Upon submission, a POST request is sent to the authentication service, including the user's username and password. The controller utilizes the AuthRequest class to send the user's credentials and captures the response in an AuthResponse object.

After processing by the authentication service, the controller receives the response. If the authentication is successful, the controller returns an entry page where the user can input new temperature data. In the event that the user is not authenticated, they are redirected back to the login page.

2. **The Entry Controller**

The Entry Controller is responsible for handling the submission of temperature entries made by the user. This web application leverages JPA and Hibernate for storing Java objects (entities) in the database. In our specific case, the entity being stored in the database is 'temperature,' with each temperature entry having an associated value.

3. **The Analytics Controller**

The Entry Controller manages user requests for statistical data. It initiates a GET request to the "show results" service and subsequently receives a list of Analytics entities. Each Analytics entity encompasses essential statistical values, including minimum temperature, maximum temperature, and average temperature. These values are transfered to a view page for the user to access only if he is authenticated.

For an in-depth look at the controller implementation, please refer to Appendix A.

Authentication Service

The enter data service is built upon the Spring REST framework for its implementation. This web application has a single controller known as the AuthenticationController. This controller fulfills the role of verifying user authentication. Upon receiving a request, the controller undertakes an assessment by examining whether both the username and password are set to "admin". If this condition is met, then the controller responds with an AuthResponse, affirming the user's authenticity. For a comprehensive review of the controller implementation, please see Appendix B.

Analytics Service

This service is built upon the Spring REST framework for its implementation. Within this web application, a lone controller, known as the AnalyticsController, takes on a pivotal role. This controller is responsible for retrieving temperature data from the MySQL table and subsequently determining key statistics such as the minimum, maximum, and average temperatures. These computed statistics are then stored in MongoDB. Notably, the AnalyticsController is invoked each time a user submits a temperature entry. Consequently, the controller performs the necessary calculations to generate valuable insights and enter them to the database.

Effortless storage and retrieval of temperature data and analytics are made possible through the use of JPA and Hibernate technologies. These tools simplify the interaction between Java objects and the database, making the process smoother. In this web application, two models are employed: the "temperature" model, which helps fetch temperature data from the database, and the "analytics" model, designed for storing temperature-related information in the database.

For a detailed overview of the controller implementation, please turn to Appendix C

Show Results Service

This service is built upon the Spring REST framework for its implementation. Within this web application, a lone controller, known as the AnalyticsController, takes on a pivotal role. This controller is responsible for retrieving temperature data from the MySQL table and subsequently determining key statistics such as the minimum, maximum, and average temperatures. These computed statistics are then stored in MongoDB. Notably, the AnalyticsController is invoked each time a user submits a temperature entry. Consequently, the controller performs the necessary calculations to generate valuable insights and enters them to the database.

Effortless storage and retrieval of temperature data and analytics are made possible through the use of JPA and Hibernate technologies. These tools simplify the interaction between Java objects and the database, making the process smoother. In this web application, two models are employed: the "temperature" model, which helps fetch temperature data from the database, and the "analytics" model, designed for storing temperature-related information in the database.

For a detailed overview of the controller implementation, please turn to Appendix C

Show Results Service

The implementation of this service relies on the Spring REST framework. In this web application, a singular controller named the ResultsController plays a crucial role. This controller's responsibility lies in retrieving analytical data from MongoDB and returning it as a list.

For a detailed overview of the controller implementation, please turn to Appendix D

CONCLUSIONS

This report illustrated the solution of the Atypon’s 2023 Containerization assignment. The assignment is successfully solved and coded using Docker’s Engine Containerization technology. The containerized microservices data collection and analytics system was successfully built using a Docker file for each service, and a Docker compose file run the system.

The system microservices successfully has performed the specific required functions; such as; Enter Data Service, Authentication Service, MySQL Database Service, Mongo Database Service, and Show Results Service. The assignment application was broken up into a series of smaller, more specialized services, each with its own database and its own business logic. Consequently, the microservices have successfully communicated with each other across common interfaces.

Finally, the Docker Engine was used successfully to deploy the application in an efficient and secure manner in the form of containers. Packaging of the software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable that runs consistently on any infrastructure was possible with using Docker’s Engine generated Containers.

V. REFERENCES

[1] What is containerization? https://www.ibm.com/topics/containerization, August 16, 2023

[2] Use containers to Build, Share and Run your applications, https://www.docker.com/resources/what-container/, August 14, 2023

[3] Dr. Fahed Jubair, UNO Game Engine Assignment, Atypon Training Program 2023.

VI. APPENDICES

Appendix A

```
From openjdk:20
copy ./target/EnterDataService-0.0.1-SNAPSHOT.jar EnterDataService-0.0.1-
SNAPSHOT.jar
CMD ["java","-jar","EnterDataService-0.0.1-SNAPSHOT.jar"]

From openjdk:20
copy ./target/AuthenticationService-0.0.1-SNAPSHOT.jar AuthenticationService-
0.0.1-SNAPSHOT.jar
CMD ["java","-jar","AuthenticationService-0.0.1-SNAPSHOT.jar"]

From openjdk:20
copy ./target/ShowResults-0.0.1-SNAPSHOT.jar ShowResults-0.0.1-SNAPSHOT.jar
CMD ["java","-jar","ShowResults-0.0.1-SNAPSHOT.jar"]

From openjdk:20
copy ./target/AnalyticsService-0.0.1-SNAPSHOT.jar AnalyticsService-0.0.1-
SNAPSHOT.jar
CMD ["java","-jar","AnalyticsService-0.0.1-SNAPSHOT.jar"]
```

Appendix B

```
@Controller
public class LoginController {

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/")
    public String loginPage() {
        return "login-page";
    }

    @PostMapping("/submit")
    public String submitPage(@RequestParam String username, @RequestParam
String password){
        AuthResponse response =
restTemplate.postForObject("http://authentication-service:8081/authenticate",
new AuthRequest(username, password), AuthResponse.class);
        if(response != null && response.isAuthenticated()){
            return "entry-page";
        }
        return "redirect:/login-page";
    }

}

@Controller
public class EntryController {

    @Autowired
    private RestTemplate restTemplate;

    @Autowired
    private TempService tempService;


    @PostMapping("/submit-entry")
    public String submitEntry(@RequestParam int temperature){
        tempService.addNewTemp(temperature);
        restTemplate.postForObject("http://analytics-service:8083/generate-
analytics", new String("A new entry has been added"), AuthResponse.class);
        return "entry-page";
    }
}
```

```
@Controller
public class AnalyticsController {

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/get-results")
    public String displayResults(Model model) {

        String url = "http://show-service:8084/results-page";
        ResponseEntity<List<Analytics>> response = restTemplate.exchange(
                url,
                HttpMethod.GET,
                null,
                new ParameterizedTypeReference<List<Analytics>>() {}
        );

        List<Analytics> analyticsList = response.getBody();
        for(Analytics analytics: analyticsList){
            System.out.println(analytics.getAvgTemp());
        }
        model.addAttribute("analyticsList", analyticsList);

        return "analytics-view";
    }
}
```

Appendix C

```
@RestController
public class AuthenticationController {

    @PostMapping("/authenticate")
    public AuthResponse authenticate (@RequestBody AuthRequest authRequest) {
        String username = authRequest.getUsername();
        String password = authRequest.getPassword();
        Boolean isAuthenticated;

        isAuthenticated = username.equals("admin") &&
password.equals("admin");

        System.out.println(username);

        return new AuthResponse(isAuthenticated);
    }
}
```

Appendix D

```
@RestController
public class AnalyticsController {

    @Autowired
    private TempService tempService;

    @Autowired
    private AnalyticsRepo analyticsRepo;


    @PostMapping("/generate-analytics")
    public void getTemperatures(){
        int temp;
        int minTemp = Integer.MAX_VALUE;
        int maxTemp = 0;
        int avgTemp;
        int sum = 0;
        List<Temperature> tempList = tempService.getAllTemps();
        for(Temperature temperature: tempList){
            temp = temperature.getTempValue();
            if(temp < minTemp){
                minTemp = temp;
            } else if(temp > maxTemp){
                maxTemp = temp;
            }
            sum += temp;
        }
        avgTemp = sum / tempList.size();
        analyticsRepo.save(new Analytics(minTemp, maxTemp, avgTemp));
        System.out.println("Latest statistics saved!");
    }
}
```

Appendix E

```
@RestController
public class ResultsController {
    @Autowired
    private AnalyticsRepo analyticsRepo;

    @GetMapping("/analytics-list")
    public List<Analytics> analyticsList() {

        List<Analytics> analyticsList = analyticsRepo.findAll();

        return analyticsList;
    }
}
```